

Semantic Analysis III

Static Semantics

EECS 483 – Lecture 14

University of Michigan

Wednesday, October 25, 2006

Guest Speaker: Simon Chen

Announcements

- ❖ Exam 1 review
 - » Monday (10/30) in class
- ❖ Exam 1
 - » Wednes (11/1) in class
- ❖ Project 2
 - » Due Monday (10/30) midnight
- ❖ Reading - None
 - » Static semantics is not covered in the book

Static Semantics

- ❖ Can describe the types used in a program
- ❖ How to describe type checking
- ❖ **Static semantics:** Formal description for the programming language
- ❖ Is to type checking:
 - » As grammar is to syntax analysis
 - » As regular expression is to lexical analysis
- ❖ Static semantics defines types for legal ASTs in the language

Type Judgments or Relations

- ❖ Static semantics = formal notation which describes type judgments:
 - » $E : T$
 - » means “E is a well-typed expression of type T”
 - » E is typable if there is some type T such that $E : T$
- ❖ Type judgment examples:
 - » $2 : \text{int}$
 - » $\text{true} : \text{bool}$
 - » $2 * (3 + 4) : \text{int}$
 - » “Hello” : string

Type Judgments for Statements

- ❖ Statements may be expressions (i.e., represent values)
- ❖ Use type judgments for statements:
 - » `if (b) 2 else 3 : int`
 - » `x == 10 : bool`
 - » `b = true, y = 2 : int` (result of comma operator is the value of the rightmost expression)
- ❖ For statements which are not expressions: use a special unit type (void or empty type)
 - » `S : unit`
 - » means “S is a well-typed statement with no result type”

Class Problem

Whats the type of the following statements?

Assume i^* are int variables, f^* are float variables

$f1 [3]$

$i = i1 [i2]$

while ($i < 10$) do $S1$

$(i ? 0) 4.0 : 1.0$

Deriving a Judgment

- ❖ Consider the judgment
 - » `if (b) 2 else 3 : int`
- ❖ What do we need to decide that this is a well-typed expression of type `int`?
 - » `b` must be a `bool` (`b : bool`)
 - » `2` must be an `int` (`2 : int`)
 - » `3` must be an `int` (`3 : int`)

Type Judgements

- ❖ Type judgment notation: $A \vdash E : T$
 - » Means “In the context A, the expression E is a well-typed expression with type T”
- ❖ Type context is a set of type bindings: $id : T$
 - » (i.e. type context = symbol table)
 - » $b: \text{bool}, x: \text{int} \vdash b: \text{bool}$
 - » $b: \text{bool}, x: \text{int} \vdash \text{if } (b) \ 2 \ \text{else } x : \text{int}$
 - » $\vdash 2 + 2 : \text{int}$

Deriving a Judgment

❖ To show

» $b: \text{bool}, x: \text{int} \vdash \text{if } (b) \ 2 \ \text{else } x : \text{int}$

❖ Need to show

» $b: \text{bool}, x: \text{int} \vdash b : \text{bool}$

» $b: \text{bool}, x: \text{int} \vdash 2 : \text{int}$

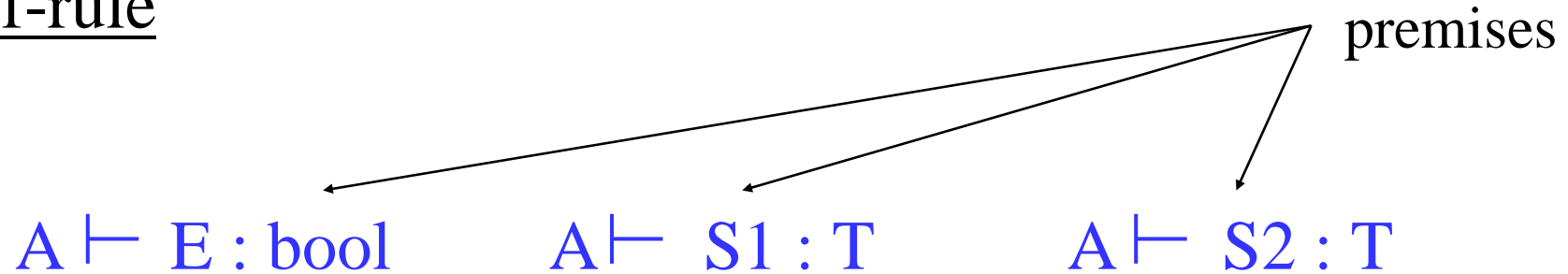
» $b: \text{bool}, x: \text{int} \vdash x : \text{int}$

General Rule

- ❖ For any environment A , expression E , statements $S1$ and $S2$, the judgement:
 - » $A \vdash \text{if } (E) S1 \text{ else } S2 : T$
- ❖ Is true if:
 - » $A \vdash E : \text{bool}$
 - » $A \vdash S1 : T$
 - » $A \vdash S2 : T$

Inference Rules

if-rule



$A \vdash \text{if } (E) S1 \text{ else } S2 : T$

conclusion

• *Read as, “if we have established the statements in the premises listed above the line, then we may derive the conclusion below the line”*

• Holds for any choice of E, S1, S2, T

Why Inference Rules?

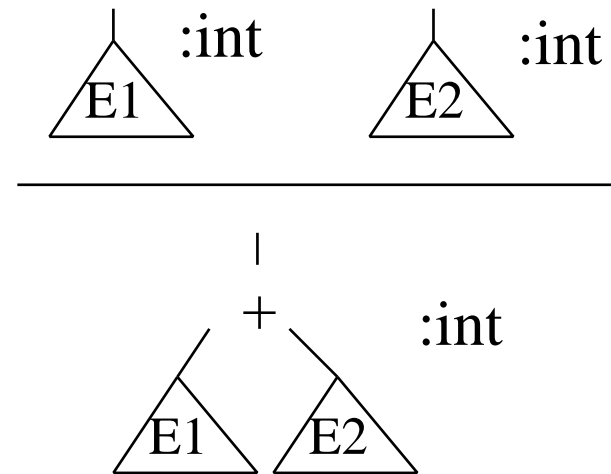
- ❖ Inference rules: compact, precise language for specifying static semantics
- ❖ Inference rules correspond directly to recursive AST traversal that implements them
- ❖ Type checking is the attempt to prove type judgments $A \vdash E : T$ true by walking backward through the rules

Meaning of Inference Rule

❖ Inference rule says:

- » Given the premises are true (with some substitutions for A, E1, E2)
- » Then, the conclusion is true (with consistent substitution)

$$\frac{A \vdash E1 : \text{int} \quad A \vdash E2 : \text{int}}{A \vdash E1 + E2 : \text{int}} \quad (+)$$



Proof Tree

- ❖ Expression is well-typed if there exists a type derivation for a type judgment
- ❖ Type derivation is a proof tree
- ❖ Example: **if $A1 = b : \text{bool}$, $x : \text{int}$, then:**

$$\frac{A1 \vdash b : \text{bool}}{A1 \vdash !b : \text{bool}} \quad \frac{A1 \vdash 2 : \text{int} \quad A1 \vdash 3 : \text{int}}{A1 \vdash 2 + 3 : \text{int}} \quad A1 \vdash x : \text{int}$$

$$b : \text{bool}, x : \text{int} \vdash \text{if } (!b) 2 + 3 \text{ else } x : \text{int}$$

More About Inference Rules

- ❖ No premises = axiom

$$\frac{}{A \vdash \text{true} : \text{bool}}$$

- ❖ A goal judgment may be proved in more than one way

$$A \vdash E1 : \text{float}$$
$$A \vdash E2 : \text{float}$$
$$\frac{}{A \vdash E1 + E2 : \text{float}}$$
$$A \vdash E1 : \text{float}$$
$$A \vdash E2 : \text{int}$$
$$\frac{}{A \vdash E1 + E2 : \text{float}}$$

- ❖ No need to search for rules to apply – they correspond to nodes in the AST

Class Problem

Given the following syntax for arithmetic expressions:

$t ::=$

true

false

if t then t else t

0

succ t

pred t

iszero t

And the following typing rules for the language:

true : bool

false : bool

$t1: \text{bool} \quad t2: T \quad t3 : T$

$\frac{}{\text{if } t1 \text{ then } t2 \text{ else } t3 : T}$

$t1 : \text{int}$

$\frac{}{\text{succ } t1 : \text{int}}$

$t1 : \text{int}$

$\frac{}{\text{pred } t1 : \text{int}}$

$t1 : \text{int}$

$\frac{}{\text{iszero } t1 : \text{bool}}$

Construct a type derivations to show

(1) if iszero 0 then 0 else pred 0 : int

(2) pred(succ(iszero(succ(pred(0)))) : int

Assignment Statements

$$\frac{\begin{array}{l} \text{id} : T \in A \\ A \vdash E : T \end{array}}{A \vdash \text{id} = E : T} \quad \text{(variable-assign)}$$

$$\frac{\begin{array}{l} A \vdash E3 : T \\ A \vdash E2 : \text{int} \\ A \vdash E1 : \text{array}[T] \end{array}}{A \vdash E1[E2] = E3 : T} \quad \text{(array-assign)}$$

If Statements

- If statement as an expression: its value is the value of the clause that is executed

$$\frac{\begin{array}{l} A \vdash E : \text{bool} \\ A \vdash S1 : T \quad A \vdash S2 : T \end{array}}{A \vdash \text{if } (E) S1 \text{ else } S2 : T} \quad \text{(if-then-else)}$$

- If with no else clause, no value, why??

$$\frac{\begin{array}{l} A \vdash E : \text{bool} \\ A \vdash S : T \end{array}}{A \vdash \text{if } (E) S : \text{unit}} \quad \text{(if-then)}$$

Class Problem

1. Show the inference rule for a while statement, while (E) S
2. Show the inference rule for a variable declaration with initializer, Type id = E
3. Show the inference rule for a question mark/colon operator, E1 ? S1 : S2

Sequence Statements

- ❖ Rule: A sequence of statements is well-typed if the first statement is well-typed, and the remaining are well-typed as well:

$$A \vdash S1 : T1$$
$$A \vdash (S2; \dots ; Sn) : Tn$$

$$A \vdash (S1; S2; \dots ; Sn) : Tn$$

(sequence)

Declarations

$A \vdash \text{id} : T [= E] : T1$

$A, \text{id} : T \vdash (S2; \dots ; Sn) : Tn$

= unit if no E

(declaration)

$A \vdash (\text{id} : T [= E]; S2; \dots ; Sn) : Tn$

Declarations add entries to the environment
(e.g., the symbol table)

Function Calls

- ❖ If expression E is a function value, it has a type $T_1 \times T_2 \times \dots \times T_n \rightarrow T_r$
- ❖ T_i are argument types; T_r is the return type
- ❖ How to type-check a function call?
 - » $E(E_1, \dots, E_n)$

$$A \vdash E : T_1 \times T_2 \times \dots \times T_n \rightarrow T_r$$
$$A \vdash E_i : T_i \quad (i \in 1 \dots n)$$

$$A \vdash E(E_1, \dots, E_n) : T_r$$

(function-call)

Function Declarations

- ❖ Consider a function declaration of the form:
 - » $\text{Tr fun } (T1 \ a1, \dots, Tn \ an) = E$
 - » Equivalent to:
 - $\text{Tr fun } (T1 \ a1, \dots, Tn \ an) \{ \text{return } E; \}$
- ❖ Type of function body S must match declared return type of function, i.e., $E : \text{Tr}$
- ❖ **But, in what type context?**

Add Arguments to Environment

- ❖ Let A be the context surrounding the function declaration.
 - » The function declaration:
 - $\text{Tr fun } (T1 \ a1, \dots, Tn \ an) = E$
 - » Is well-formed if
 - $A, a1 : T1, \dots, an : Tn \vdash E : \text{Tr}$
- ❖ What about recursion?
 - » Need: $\text{fun: } T1 \times T2 \times \dots \times Tn \rightarrow \text{Tr} \in A$

Class Problem

Recursive function – factorial

```
int fact(int x) = if (x == 0) 1 else x * fact(x-1);
```

Is this well-formed?, if so construct the type derivation

Mutual Recursion

❖ Example

» `int f(int x) = g(x) + 1;`

» `int g(int x) = f(x) - 1;`

❖ Need environment containing at least

- $f: \text{int} \rightarrow \text{int}, g: \text{int} \rightarrow \text{int}$
- when checking both f and g

❖ Two-pass approach:

- » Scan top level of AST picking up all function signatures and creating an environment binding all global identifiers
- » Type-check each function individually using this global environment

How to Check Return?

$$\frac{A \vdash E : T}{A \vdash \text{return } E : \text{unit}} \quad (\text{return})$$

- ❖ A return statement produces no value for its containing context to use
- ❖ Does not return control to containing context
- ❖ Suppose we use type unit ...
 - » Then how to make sure the return type of the current function is T??

Put Return in the Symbol Table

- ❖ Add a special entry $\{\text{return_fun} : T\}$ when we start checking the function “fun”, look up this entry when we hit a return statement
- ❖ To check $\text{Tr fun } (T1 \ a1, \dots, Tn \ an) \{ S \}$ in environment A , need to check:

$A, a1 : T1, \dots, an : Tn, \text{return_fun} : \text{Tr} \vdash A : \text{Tr}$

$$\frac{A \vdash E : T \quad \text{return_fun} : T \in A}{A \vdash \text{return } E : \text{unit}} \quad (\text{return})$$

Static Semantics Summary

- ❖ Static semantics = formal specification of type-checking rules
- ❖ Concise form of static semantics: typing rules expressed as inference rules
- ❖ Expression and statements are well-formed (or well-typed) if a typing derivation (proof tree) can be constructed using the inference rules

Review of Semantic Analysis

- ❖ Check errors not detected by lexical or syntax analysis
- ❖ Scope errors
 - » Variables not defined
 - » Multiple declarations
- ❖ Type errors
 - » Assignment of values of different types
 - » Invocation of functions with different number of parameters or parameters of incorrect type
 - » Incorrect use of return statements

Other Forms of Semantic Analysis

- ❖ One more category that we have not discussed
- ❖ Control flow errors
 - » Must verify that a **break** or **continue** statements are always enclosed by a while (or for) stmt
 - » Java: must verify that a **break X** statement is enclosed by a for loop with label X
 - » **Goto labels** exist in the proper function
 - » Can easily check control-flow errors by recursively traversing the AST

Where We Are...

